A protocol for a message system for the tiles of the heptagrid, in the hyperbolic plane

Maurice Margenstern

professor emeritus of Université Paul Verlaine − Metz
LITA, EA 3097, UFR-MIM, and CNRS, LORIA
Campus du Saulcy, 57045 Metz, Cédex, France
e-mail: margens@univ-metz.fr

Abstract. This paper introduces a communication system for the tiles of the heptagrid, a tiling of the hyperbolic plane. The method can be extended to other tilings of this plane. The paper focuses on an actual implementation at the programming stage with a short account of two experiments.

Keywords: hyperbolic tilings, cellular automata, applications.

1 Introduction

In this paper, we present a protocol to manage communications between tiles of a tiling of the hyperbolic plane. Why the hyperbolic plane? Because the geometry of this space allows to implement there any tree structure. The reason for that is that the simplest tilings defined in this plane are spanned by a tree. We refer to the author's papers and books on this topic, see [5,6]. Tree structures are already intensively used in computer science, especially by operating systems. But the fact that trees are naturally embedded in this geometrical space, especially on tilings living there, was never used. This paper proposes to take advantage of this property.

In Section 2, we remind what is needed of hyperbolic geometry in order the reader could understand the content of the paper.

In Section 2, we remind what is needed of hyperbolic geometry in order the reader could understand the content of the paper. In Section 3, we sketchily describe the navigation technique with a new aspect which was not used in [7]. In Section 4, we present the protocol which allows us to improve the system briefly mentioned in [7]. In Section 5, we give an account of the simulation program and in Section 6 we present the experiment which was performed by running the simulation program. In Section 7, we conclude with further possible development of the scenario implemented by the protocol.

2 Hyperbolic geometry and its tilings

Our first sub-section reminds the Poincaré's disc model which allows us to have a partial visualization of the hyperbolic plane. Our second sub-section defines the simplest tilings which can there be defined, allowing us to construct **grids**. In our third sub-section, we focus on the tiling on which our proposal is based, the tiling $\{7, 3\}$ of the hyperbolic plane which we call the **heptagrid**.

2.1 Hyperbolic geometry

Hyperbolic geometry appeared in the first half of the 19^{th} century, proving the independence of the parallel axiom of Euclidean geometry. This was the end of a search during two thousand years in order to prove that the well known axiom about parallels in Euclid's treatise is a consequent of his other axioms. The search is by itself a very interesting story, full of deep teachings of high price for the philosophy of sciences, we recommend the interested reader to have a look at [1]. Hyperbolic geometry was the first issue of the notion of axiomatic independence. But also, it raised the first doubts on the absolute power of our abstract mind. It opened the way to the foundational works of mathematical logics, so that computer science, the daughter of logics, appears to be a grandchild of hyperbolic geometry. The paper hopes to show that this kind of relations holds not only on a philosophical ground.

The search failed. This was proved in the second third of the 19^{th} century by the discovery of hyperbolic geometry. In this new geometry, all the non-parallel axioms of Euclidean old and, from a point A out of a line ℓ and in the plane defined by ℓ and A there are two parallels to ℓ passing through A. Around forty years after the discovery, models of the new geometry in the Euclidean one were found. Presently, we turn to the most popular model of the hyperbolic plane nowadays, Poincaré's model.

2.2 The Poincaré's disc model

This model is represented by Figure 1. In the model, the points of the hyperbolic plane are represented by the points of an open disc fixed in advance called the **unit disc**. The circle which is the boundary of the disc is called the set of **points at infinity** and we denote it by ∂U. These points do not belong to the hyperbolic plane, but they play an important role in this geometry. The lines of the hyperbolic plane are represented by the trace in the disc of its diameters or by the trace in the disc of circles which are orthogonal to ∂U. It is not difficult to see that the lines are also characterized in the models by their points at infinity as diameters of the disc and circles orthogonal to ∂U meet ∂U twice exactly.

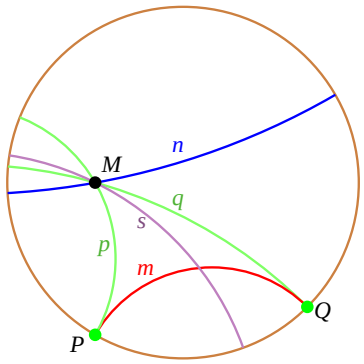

Figure 1*: Poincaré's disc model. We can see that in the model, both lines p and q pass through A and that they are parallel to ℓ: p and q touch m in the model at P and Q, points at infinity.*

The figure also shows us an important feature of the new plane: outside parallel and secant lines to a given line ℓ, passing through a point A not on ℓ, there are also lines passing through A which do not cut ℓ, neither inside the disc, nor on ∂U and nor outside the disc: they are called **non-secant** lines. A last but not least property of the new plane is that there are no similarity: a figure of this plane cannot be re sized at another scale. Resizing necessarily changes the shape of the figure.

2.3 The tilings $\{p, q\}$ of the hyperbolic plane

Consider the following process. We start from a convex regular polygon P. We replicate P in its sides and, recursively, the images in their sides. If we cover the plane without overlapping, then we say that P **tiles the plane by tessellation**. We shall often say for short that P **tiles the plane**, here, the hyperbolic plane. A theorem proved by Poincaré in 1882 tells us that if P has p sides and if its interior angle is $\dfrac{2\pi}{q}$, then P tiles the hyperbolic plane, provided that:

$$\frac{1}{p} + \frac{1}{q} < \frac{1}{2}. \tag{1}$$

The numbers p and q characterize the tiling which is denoted $\{p, q\}$ and the condition says that the considered polygons live in the hyperbolic plane. This inequality entails that there are infinitely many tessellations of the hyperbolic plane. The tessellation attached to p and q satisfying the inequality is denoted by $\{p, q\}$.

Note that we find the well known Euclidean tessellations if we replace $<$ by $=$ in the above expression. We get, in this way, $\{4, 4\}$ for the square, $\{3, 6\}$ for the equilateral triangle and $\{6, 3\}$ for the regular hexagon.

In [5,6], the author provides the reader with a uniform treatment of these tessellations. The basic feature is that each tessellations is spanned by a tree whose structure obeys well defined properties. We shall illustrate these points in our next sub-section where we focus on a particular tessellation: the tiling $\{7, 3\}$ of the hyperbolic plane which we call the **heptagrid**.

2.4 The heptagrid

The heptagrid is illustrated by Figure 2. The figure is very symmetric, but at this stage, it is difficult to identify each tile of the figure, especially for the tiles which are outside the first two rings around the central cell.

The tiles look very much the hexagonal tiles of the corresponding tessellation of the Euclidean plane,but the global organization is very different. It seems to us that this figure indicates that we need something to locate the tiles. We turn to this point in Section 3

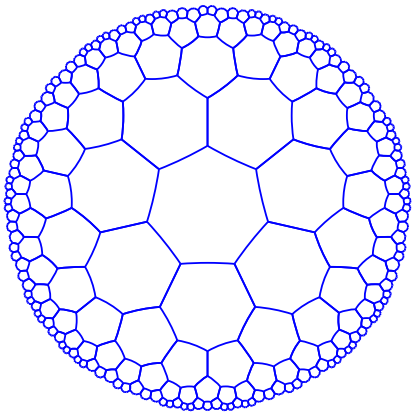

Figure 2 *The heptagrid: an illustrative representation.*

3 Navigation in the heptagrid

We have seen on the heptagrid that it is not easy to locate the tiles of this tiling, especially as we go further and further from the centre of the disc.

This is the point to draw the attention of the reader on two points. The disc model may be misleading if we forget that we are in the Euclidean plane and that some symmetries of the figure have no counter part in hyperbolic geometry. As an example, the central tile seems to play an important role, in particular its centre. However, the hyperbolic plane has no central point as well as the Euclidean plane has no such point. We decide to fix an origin when we define coordinates in the Euclidean plane. This is the same here. The central tile is simply a tile which we decide to be the origin of our coordinate system. The

second point is that Poincaré's disc model gives us a local picture only. We see the immediate neighbourhood of a point which we decided to place at the centre of the disc. The models looks like a small window on the hyperbolic plane whose central part only is well observable. From this lens effect, we conclude that walking on the hyperbolic plane, we are in the situation of the pilot of a plane flying with instruments only.

And so, what are our instruments?

The first two pictures of Figure 3 represent them. On the left-hand side of the figure, we can seen the **mid-point lines** defined by the following property: the line which joins the mid-points of two consecutive sides of a heptagon cuts a heptagon of the tiling at the mid-point of a side only. If we consider two rays issued from the meeting point of two secant mid-point lines and defined by the smallest angle, we can define a set of tiles for which all vertices but possibly one or two lie inside this angle. We call this the restriction of the tiling to a sector. The remarkable property is that the restriction of the tiling to a sector is spanned by a tree, the tree which is illustrated by the right-hand side of the figure. In the middle of the figure, we can see that the whole tiling can exactly be split into a central cell and seven sectors dispatched around the central tile.

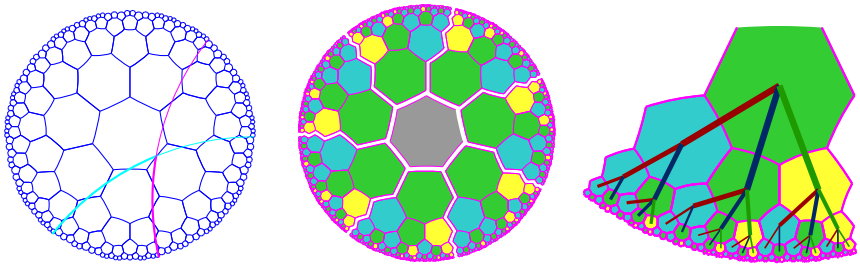

Figure 3 *Left-hand side: the mid-point lines. This tool which shows how a sector spanned by the tree is defined in the heptagrid.*

Middle: first part of the splitting, around a central tile, fixed in advance, seven sectors. Each of them is spanned by the tree represented in the right-hand side.

Right-hand side: the tree which spans the tiling.

The tree which spans the tiling can be generated by very simple rules indicating the exact connection between a node and its sons. There are two kind of nodes, the **black** and the **white** ones. The rules are, in self-explaining notations:

$$B \to BW, \qquad W \to BWW \tag{2}$$

From these rules, it can be proved that the number of nodes which are on the level n of the tree is f_{2n+1} where $\{f_n\}_{n\in\mathbb{N}}$ is the Fibonacci sequence defined by $f_0 = f_1 = 1$ and the induction equation $f_{n+2} = f_{n+1} + f_n$. For this reason, the tree is called the **Fibonacci tree**.

It is known that natural numbers can be represented as sums of distinct terms of the Fibonacci sequence: $n = \sum_{i=0}^{k} a_i f_i$, with $a_i \in \{0,1\}$, $a_k \neq 0$ if $n \neq 0$. This representation is not unique, but it can be made unique by requiring k to be maximal in the above representation. This is the **greedy Fibonacci** representation of n. It is characterized by the fact that considering the a_i's in their order as a word on $\{0,1\}^*$, there are no contiguous 1's in the word.

Presently, number the nodes of the tree level by level and, on each level, from the left to the right, starting from the root which receives 1 as its number. Next, call **coordinate** of a node ν the greedy Fibonacci representation of its number. Then the coordinates of the nodes of the tree have this striking property. If $[\nu]$ is the coordinate of ν, among the sons of ν there is a single node whose coordinate is $[\nu]00$, which is called the **preferred son**. Also, we can rewrite (2) as (3)

$$B \to \overline{B}W, \qquad W \to B\overline{W}W \tag{3}$$

where the bar indicates the position of the preferred son.

A **path** between a tile A and a tile B is a sequence $\{T_i\}_{i\in[0..n]}$ with $T_0 = A$, $T_n = B$ and, for each $i \in [1..n]$, T_{i-1} and T_i have a common side. We say that n is the length of the path and we note that from this definition, the path is oriented: its **source** is the tile T_0 and its **target** is the tile T_n. We also say that the path goes from A to B. Clearly, a path from B to A is obtained by reversing the numbering of the T_i's defining the path from A to B.

For complexity reasons, it is convenient to take a **shortest** path between A and B: it is a path between A and B whose length is minimal. Note that in general there is no unique shortest path but, by the very definition, such a path exists. An important particular case is when both tiles are on the same branch of a tree: this part of the branch is a shortest path between the tiles.

From the properties of the preferred son, we obtain:

Theorem 1 (see[3,5]) *There is an algorithm which computes the path from a node of the Fibonacci tree to the root from the coordinate of the node which is linear in the size of the coordinate.*

From this algorithm, it is easy to compute a path between two nodes which is most often almost a shortest path between the nodes: first, go from the nodes to the root and connect the two paths at the root. Moreover, the computation is linear in the size of the two coordinates. However, in certain situations, this is not the shortest way. Now, in [6], we proved a refinement of Theorem 1. In order to state it properly, we have to define coordinates for the tiles of the heptagrid.

To do this, we look at the middle picture of Figure 3. We increasingly number the sectors from 1 up to 7 by counter-clockwise turning around the central cell, fixing the sector which receives number 1 once and for all. Now, the coordinate of a tile is defined by 0 for the central tile and, for any other tile T, by the couple (σ, ν) where $\sigma \in \{1..7\}$ defines the sector which contains the tile and where ν is the coordinate of T in the tree which spans the sector.

Now, we can state the result:

Theorem 2 (see[6]) *There is an algorithm which computes a shortest path between two tiles of the heptagrid which is linear in the size of the coordinates of the tiles.*

Another consequence of Theorem 1 is the computation of the coordinates of the neighbours of a tile, where a neighbour of the tile T is a tile N which shares a side with T. Clearly, T is also a neighbour of T and we shall often say that N and T are neighbours.

The computation of the coordinates of the neighbours relies on two functions of n, the number of a tile: $f(n)$ which is the number of the father of the tile and $\sigma(n)$ which is the number of the preferred son of the tile. According to the previous notations, it is also interesting to consider the function $[f]$ such that $[f]([n]) = [f(n)]$ and the function $[\sigma]$ such that $[\sigma]([n]) = [\sigma(n)]$. From the proof of Theorem 1, there is an algorithm which allows to compute $[f]$ and $[\sigma]$ which is linear in the size of $[n]$. However, note that in the computation of the path the application of the algorithm at one step is in fact a constant. With the help of this function, the coordinates of the neighbours of a tile are given by Tables 1 and 2. In the table, the neighbours of a tile T are increasingly numbered from 1 to 7 while counter-clockwise turning around T. Neighbour 1 is the father of T. The father of a root is the central tile. For the central tile, the neighbour i is the tile associated to the root of the tree in the sector i. If i is the number of a neighbour of T, we say that the side shared by T and this neighbour is also numbered by i.

Table 1: The numbers of the neighbours for a tile ν which is inside the tree. The tile may be black or white.

τ	black	white
1	$f(\nu)$	$f(\nu)$
2	$f(\nu)-1$	$\nu-1$
3	$\nu-1$	$\sigma(\nu)-1$
4	$\sigma(\nu)$	$\sigma(\nu)$
5	$\sigma(\nu)+1$	$\sigma(\nu)+1$
6	$\sigma(\nu)+2$	$\sigma(\nu)+2$
7	$\nu+1$	$\nu+1$

Table 1 considers the general case, when the node associated to the tile is inside the tree. This means that the corresponding node always has its father in the tree and that all its neighbours are also in the tree. In Table 2, we have the exceptional cases. The nodes on the leftmost branch, the root excepted, and those which are on the rightmost branch, the root excepted. The nodes of the rightmost branch are white and the root is also white. The nodes on the leftmost branch, the root excepted, are black.

It remains to indicate that in the case of a heptagon H which is on the left- or the rightmost branch, it is easy to define the number of the sector to which belongs the neighbours which do not belong to the tree of H. Indeed, let σ be the number of the sector in which H lies. If H is a black node, its neighbours 2 and 3 are in the sector $\sigma \ominus 1$, where $\sigma \ominus 1 = \sigma - 1$ when $i > 1$ and $1 \ominus 1 = 7$. If H is a white node, its neighbours 6 and 7 are in the sector $\sigma \oplus 1$ with $\sigma \oplus 1 = \sigma + 1$ when $i < 7$ and $7 \oplus 1 = 1$. Note that for the root of the sector σ, its neighbour 2 is in the sector $\sigma \ominus 1$ and its neighbour 1 is the central cell which is outside all the sectors.

Table 2: The numbers of the neighbours for a tile ν which is either the root of the tree, or which belongs to an extremal branch the leftmost or the rightmost ones. The numbers are given in the columns `root`, `left` and `right` respectively.

τ	left	right	root
1	$f(\nu)$	$f(\nu)$	0
2	$\nu-1$	$\nu-1$	1
3	$\sigma(\nu)-1$	$\sigma(\nu)-1$	$\sigma(\nu)-1$
4	$\sigma(\nu)$	$\sigma(\nu)$	$\sigma(\nu)$
5	$\sigma(\nu)+1$	$\sigma(\nu)+1$	$\sigma(\nu)+1$
6	$\sigma(\nu)+2$	$\nu+1$	$\nu+1$
7	$\nu+1$	$f(\nu)+1$	1

Both Theorems 1 and 2 give an algorithm to compute new coordinates if we change the place of the central tile to another tile. In Section 5, we briefly give an explicit pseudo code implementing an algorithm satisfying Theorem 2.

The computation of the functions f and σ used in the computation of the coordinates of the neighbours of a tile are also used for this purpose and based on these considerations and on the previous results we have that:

Theorem 3 (see [6]) *Consider a system of coordinate and a tile T. Assume that we take T as the central tile and that we fix its new side 1. There is an algorithm which, for each tile T' computes the coordinates of T' in the new system in linear time in the size of the coordinate in the initially given system.*

It is important to remark that in order to obtain the linearity of the algorithm, we do not compute the functions f and σ but $[f]$ and $[\sigma]$ applied to $[n]$. This means that in order to compute $\sigma(n)-1$ for instance, we need an algorithm for computing $[m-1]$ from $[m]$. A similar remark holds for $[m+1]$. The needed algorithms can be found in [5].

4 The communication protocol

In Section 1, we mentioned that in [7], we already proposed a communication protocol for the tiles of the heptagrid. This protocol was based on a specific system of coordinates, inherited from [4,6]. For the convenience of the reader, we briefly describe this system in Sub-section 4.1. Then, in Sub-section 4.2 we define the new protocol.

4.1 Absolute and relative systems

The **absolute** system is based on a numbering of the sides tiles of the heptagrid. For each tile, we number the sides from 1 to 7 in this order while counter-clockwise turning around the tile. Now, how to fix side 1? We again take the situation of the left-hand side of Figure 3: a central cell surrounded by seven sectors, each one spanned by a copy of the Fibonacci tree. Now, side 1 is fixed once and for all for the central tile. For the other tiles, side 1 is the side shared by the tile and its father, considering that the central cell is the father of the root for each copy of the Fibonacci tree spanning the sectors.

Now, a side always belongs to two tiles and so it receives two numbers. This is why this numbering is called **local**. However, the association between both numbers is not arbitrary. There is a correspondence between them although it is not one to one. When one number is known, the status of the tile and the fact that the corresponding node whether lies or not on an extremal branch of the tree are also needed to determine the other number. This correspondence is given by Table 3 and Table 4 lists all the couples used by the sides: note that we are far from using all possible couples.

The absolute system consists in first fixing the local numbering once and for all. Table 3 will still be used to determine the two numbers of a side of a heptagon.

Table 3: Correspondence between the numbers of a side shared by two heptagons, H and K. Note that if H is white, the other number of side 1 may be 4 or 5 when K is white and that it is always 5 when K is black.

black H		white H	
in H	in K	in H	in K
1	3^{wK},4^{bK}	1	4^{wK}, 5
2	6	2	7
3	7	3	1
4	1	4	1
5	1	5	1
6	2	6	2
7	2	7	2^{wK}, 3^{bK}

Next, we remark that the local numbering gives a way to encode a path between two tiles A and B. Let $\{T_i\}_{i\in[0..n]}$ be a shortest path from A to B and denote by s_i the side shared by T_i and T_{i+1} for $i \in [0..n-1]$. Let a_i be the number of s_i in T_i and b_i be its number in T_{i+1}. Then we say that the sequence $\{(a_i, b_i)\}_{i\in[0..n-1]}$ is an **address** of B **from** A. The reverse sequence gives an address of A from B. However, on the just above sequence, we do not know from the sequence itself that (a_n, b_n) is the last side. In order to do this, we change a bit the association of the numbers: for T_i belonging to the path, we denote by en_i the side shared with T_{i-1} and by ex_i the side shared with T_{i+1}. For T_0, as en_0 cannot be defined as the number of a neighbour of T_0, we put $en_0 = 0$. This time we say that the sequence $\{(en_i, ex_i)\}_{i\in[O..n]}$ is the **coordinate** of B from A. Similarly, the sequence $\{(ex_{n-i}, en_{n-i})\}_{i\in[0..n]}$ is the coordinate of A from B. And so, the correspondence between the address and the coordinate is easy: $a_i = ex_i$ and $b_i = en_{i+1}$ for $i \in [0..n-1]$ which contains the definitions of ex_0 and en_n.

Now, how to define a shortest path between A and B?

There are two ways: the first way is given by Theorem 2. We apply the algorithm defined in Section 5 in order to find the coordinate of A from B.

The second way consists in the following. If A sends messages to every tile, it considers itself as the central tile, taking its own number 1 as the number 1 of the central cell. Remember that all tiles have the same size, the same shape and the same area, this is why each tile may feels 'equal' to the others. When it sends the message to its neighbours, it also send them the information that it is the central cell and it sends $(0, i)$ to its neighbour i. And so, the neighbour receives its coordinate from A. By induction, we assume that each tile T which receives the message from A also receives its coordinate from A and its status in the relative tree to A in which T is. From this information, and as T knows from which neighbour it receives the message, T know which of its neighbours are its relative sons and so, it can append the element (en_T, ex_T) to the address it conveys to the corresponding son together with the relative status of the son. And so, we proved that each tile receiving a message from A also receives its address from A and its relative status with respect to A. In fact, we have an implementation of the local numbering attached to A as a central tile. This local numbering is called the **relative** system. Now, from its coordinate from A, T may computes the coordinate of A from T in a linear time in the size of the coordinate, as follows from what we already have noticed. And so, if T wishes to reply to the message sent by A it can do it easily. Moreover, from the properties we have seen in Section 3, we can see that, proceeding in the just described way, a public message is sent to every tile once exactly, which is an important feature.

4.2 The protocol

We have now the tools to describe the protocol of communication between the tiles.

For this protocol, we distinguish two types of messages, **public** ones and **private** ones. By definition, a public message is a message sent by a tile to all

the other tiles. A private message is a message sent by a tile to a single other one. This distinction belongs to the sender of the message. In this protocol, we assume that we have a global clock defining a discrete time and that a message leaving a tile T at time t can reach only a neighbour of T at time $t+1$. We say that the maximal speed for a message is 1.

The public message makes use of the relative system of the sender. However, in the coordinates which are constructed by the tiles which relay the message, the numbers en_i and ex_i computed by the relaying tile are defined according to the absolute system as the tile does not know where is the sender and as its own local numbering is defined by the absolute system.

A private message is either a reply to a message, either public or private, or a message sent to a single tile according to the following procedure. Each tile T has a direct access to a the system. Given the coordinates of a tile N as defined in Section 3, the central cell being that of the absolute system, the managing system gives to T a shortest path from T to N which is a coordinate of N from T. And so, a private message is defined by the fact that it has the address of the receiver.

Table 4*: The pairs (i, j) of numbers of a side of a heptagon. It is assumed that the first number denotes the side*

1	(1,3)
	(1,4)
	(1,5)
2	(2,6)
	(2,7)
3	(3,7)
	(3,1)

4	(4,1)
5	(5,1)
6	(6,2)
7	(7,2)
	(7,3)

In order to deliver the information efficiently, a private message from A to B stores the coordinate of B as two stacks a and r. The stack a is for the direct run from A to B, the stack b is for the way back. Each tile T on the path conveys the message to the next one N on the path, in the direction from A to B. To perform this, T reads the top of a, say (en, ex). It knows that ex is the number of N from itself. Just before sending N the message and the stacks, T pops the top (en, ex) of a and pushes (ex, en) on the top of r. In this way, T knows that it is the receiver if $ex = 0$. When this is the case, T pops the top (en, ex) of a, pushes (ex, en) on r but does nothing else. Note that at this moment a is empty. When T is ready to answer, it exchange a and r and so, the same process allows the message to reach A together with a coordinate of B from A. This is illustrated by Figure 4 on a toy example.

It is easy to see that this process is linear in time with respect to the coordinate of the receiver, assuming that all messages travel at maximal speed 1.

Now, we shall see that in the process of a public message, the computation by the relaying tile of the new information which it has to append to the message is easily computed. Here two, we have two stacks, but the stack r is always empty.

Consider a relaying tile T. Let (en_0, ex_0) be the top of the stack a. By construction, ex_0 is the side of the neighbour N through which T received the message. In order to facilitate the computation, T also receives the number en_1 in T of the side numbered ex_0 in N. Let s be the number of the relative son of T in the relative tree. We know that $s \in [3..5]$ if T is white in the relative tree and that $s \in [4, 5]$ if T is black in the relative tree. This index corresponds to a position of the father at the absolute index 1. Now, the absolute index ex_1 of the son defined by s is given by:

$$ex_1 = 1 + ((en_1 - 1) + s - 1) \bmod 7, \tag{4}$$

Once ex_1 is known, the other absolute number of the side defined by ex_1, say en_2, may be determined by Table 4. Now, we know that when $ex_1 \in \{1, 2, 3, 7\}$, en_2 is not uniquely defined. The value of en_2 depends on the absolute status of T. The simplest solution is to assume that all pairs (i, j) for $i \in \{1..7\}$ are known for each tile T in a table *output* which is a sub-table of Table 4, see Section 5 for the implementation of this important point. Then we have:

$$en_2 = output(ex_1).$$

For implementation, note that the root of the relative sector 1 for a sender is its absolute neighbour 1. Also note that formula (4) is different from the formula given in [6]. In formula (4), we do not need to know the relative status of the tile, contrarily to the formula used in [6]. This is automatically given by the relative indices used for the relative sons.

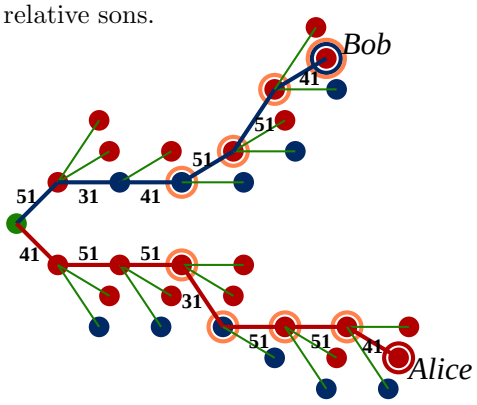

Figure 4 *Illustration of the protocol of communication between tiles of the pentagrid or of the heptagrid. The circled tiles indicate a shortest path from Alice to Bob. Note the numbering of the sides shared by two tiles according to the definition of this sub-section.*

However, we cannot assume that all tiles send messages at any time. This would not be realistic. Also, we cannot assume that public messages are sent for ever to cover the whole plane which would also not be realistic. Indeed, in case

of public messages sent without stopping, the number of messages at any tile at each time would increase to infinity at an exponential rate with time.

In order to limit the scope of a public message, we define a **radius** of its propagation. This means that if a public message is sent from A, it will reach any tile whose distance from A is at most the radius. The distance between two tiles A and B is the length of a shortest path between A and B. Of course, the message could also bring with it the delay which could be decremented by 1 each time it reaches a new tile, and the message would destroy itself when the delay would be 0. The defect of this solution is that we have to transport the delay to all tiles within the radius and that at each time, we have to perform this decrementing at each tile.

There is another solution. When A sends a public message with radius r, the message is not sent at maximal speed 1 but at a speed $\frac{1}{2}$. As we have a global clock, we shall distinguish between odd and even times. A public message travels at odd times and remains on the tile at even times. Consider the message μ sent from A at time 1. Now, A must remember μ and this is implemented as another message μ_e which is not sent immediately. The new message has the minimal information. It has to destroy μ and only μ. To this purpose, all messages are identified by a unique number given by the system. And so, μ_e contains the number of μ. Next, A keeps μ_e during r tops of the clock. In fact, μ_e also keeps a delay which is r when μ_e is created and which is decremented at each top of the clock. When the delay is 0, μ_e is sent to all tiles, according to the same process as a public message, but at speed 1. Also, another important difference is that μ_e does not need to transport a stack as it is not sent in order to get any reply. As μ_e travels at speed 1, at time $2r$, it reaches μ and it destroys it. Later on we shall say that μ_e is an **erasing** message.

With this, we completed the description of the process concerning public messages. Private messages always travel at speed 1.

A last point for the simulation: as a tile does not send a message at any time, we have to decide when it sends a message. For this purpose, we use a Poisson generator, both for the decision of sending a message and, in the case of a public message, for defining the radius of the propagation of the message. For each parameter, we use a different coefficient for the generator. We shall see the values in Section 6.

5 The simulation program

As usual for the implementation of a theoretical model, the simulation program results from many choices decided by the programmer for the implementation of various features, in particular, the structure representing the space of the simulation, we shall see this point in Subsection 5.1. Subsection 5.2 is devoted to auxiliary computations connected with the implementation of the basic algorithms. Subsection 5.3 describes the exact implementation of the scenario described in Section 4.2. The simulation program was written in $ADA95$. When this is the case, we mention facilities given by the programming language.

5.1 Data structures

In a first stage of the experiments described in Section 6, the heptagrid was implemented as a table `space` with two entries: one in 0..7 and the other in 0..`maxsize`, where `maxsize` is the number of tiles of a sector represented by the simulation, the sector i, with $i \in [1..7]$ being represented by `space(i,*)`. From section 3, it is not difficult to compute that up to the level n, the number of tiles in a sector is $f_{2n+2}-1$.

The elements of the table are a small table of 8 records indexed from 0 to 8. Record 0 represents the tile `space`(i, j): it is the tile T of coordinate (i, j) with $i \in [1..7]$ and $j \in [1..$`maxsize`$]$. Record v, with $v \in [1..7]$ represents the neighbour v of T. The fields of the record contain the information needed to minimize the computation time. Except the coordinate of the tile and of its neighbours, the record is assumed to contain an information of constant size.

Presently, in order to overcome a quick overflow by the manipulation of the table, the space of the experiment is implemented by stacks. There is a basic table `space` with a single entry in 0..7 which represents the central cell and the roots of the Fibonacci trees. Each element of the table is a pointer, `addresstile` which points at a tile. The pointer in `space(0)` points at the central cell, `space(`i`)`, with $i \in 1..7$ points at the root of the sector i. The tiles themselves are represented by a record whose fields are:

`num`: the number of the node in its tree,
`sect`: the number of the sector,
`neighbour`: a table of seven pointers, `neighbour(`i`)` pointing at the neighbour i of the tile,
`associate`: a table of seven numbers, the absolute number of side i in the neighbour i,
`branch`, with the self-explaining values about the position of the tile with respect to the borders of the sector: `left`, `right` `middle`, `root` and `centre`,
`status`, the status of the tile: `central`, `white` or `black`,
`outer`, boolean: says if the considered neighbour is outside the simulation space,
`border`, boolean: says if the tile is on the border of the simulation space,
`message_stack0`, `message_stack1`: two pointers on the stack of messages,
`last_message0`, `last_message1`: two pointers at the last element of the stack, *i.e.* the most recent message.

The stack of messages contains what is needed for the communication system. Each element of the stack holds a record with the following fields:

`next` : pointer for handling the stack,
`relative_father`, an integer in 1..7, 0 if not defined,
`relative_status`, with the values: `black`, `white`, `centre`,
`number`, an integer: the number given to the message,

`the_type`, with the values: `public`, `private`, `erasing`,
`wait`, an integer: the counter for an erasing message,
`direct, wayback`, pointers of messages.

Now, why two pointers at the stack of messages?

This is to perform the simulation in the following way. At each tile, we represent the stack of messages by two disjoint stacks: `stack0`, accessed through `message_stack0` and `last_message0`, and `stack1`, accessed through the pointers `message_stack1` and `last_message1`. We consider that `stack0` represents the stack at the tile at the time t while `stack1` represents the stack at the same tile but at the time $t+1$. The disjunction of `stack0` with respect to `stack1` avoid any confusion of pointers. And this disjunction is needed as a tile possibly receives contributions from its neighbours and this is performed sequentially by the simulation, whence the distinction between the two configurations of the same stack at the time t and at the time $t+1$: the same tile receives contribution from its neighbours at different steps of the sequential process within the steps of computation which represent the turning from the time t to the time $t+1$.

5.2 Auxiliary computations

The computation of the shortest path is given by a simple algorithm which hides more involved computations, although they remain within a linear estimate with a small coefficient, see 1. Among the shortest paths from the tile A to the tile B, there are two extremal ones: if we consider the A as the central cell, B lies in a sector i and the the path from A to B which passes through the root of the sector and going along the branch of the tree from the root to B is one of these extremal paths: there is no other shortest path on the left-hand side of this path while looking at B and the root with B below the root. If we start from B as the central cell, we get the other extremal path: no shortest to the right-hand side of this one. Now, when we have at our disposal a shortest path π, say from A to B, it is possible to define the leftmost shortest path from A to B. This path is computed thanks to the procedure `measure` which computes the distance between two paths. It also relies on a function `pathroot` which computes the path from a node to the root from the coordinate of the node. It uses a function `leftmost` which computes the leftmost shortest path when it is given a shortest path.

The function `pathroot` is a significantly improved version of the algorithm given in [6], giving a much simpler proof of Theorem 1. The key point was to notice that there is a kind of propagation of the carry when for the first time to contiguous 1's are detected: test in the `else`-branch of the main `if` of the loop.

When starting the execution of function `shortest`, both paths start from the central cell as indicated by the instructions which initialize `Lcursor` and `Rcursor`. As suggested by the identifiers, `Lcursor` points at the leftmost tiles between `tile1` and `tile2`. The function `theleftmost` looks whether the points belong to the same sector of not. If not, the absolute difference between the indices of the sectors allows us to know the leftmost tile. If they are in the same

sector, then the function follows both paths from the root to the nodes until the paths diverge as we assume that the tiles are distinct. At the tile which is the point where both paths diverge, it is easy to determine which tile is on the left-hand side of the other.

Algorithm 1 *The computation of the shortest path. Given* `tile1` *and* `tile2` *with the assumption that* `tile1` $\neq$ `tile2`, `tile1` $\neq \emptyset$ *and* `tile2` $\neq \emptyset$ *as well.*

```
Ltile := theleftmost(tile1,tile2);
if tile1 = Ltile
  then Rtile := tile2;
  else Rtile := tile1;
end if;
Lcursor := chain_translation(pathroot(Ltile));
Rcursor := chain_translation(leftmost(pathroot(Rtile)));
loop
   measure(distance,Lcursor,RcursoR);
   if Lcursor.next = null then exit; end if;
   if Rcursor.next = null then exit; end if;
   exit when distance > 1;
   Lcursor := Lcursor.next;
   Rcursor := Rcursor.next;
end loop;
-- we go out of the loop when distance ≥ 2
-- the distance is between Lcursor.next et Rcursor.next
ladresse := connect (Lcursor,Rcursor);
return ladresse;
```

When the leftmost tile is determined, then the function first computes the path from `Ltile` to the central cell and then the leftmost path from `Rtile` to the central cell. In this way, we already know that the distance between the paths is at most 1 as long as possible. In the loop which starts form the central tile, the distance between the tiles of the path at the current stage is measured by the procedure `measure` which updates `distance` which outputs the computed distance. If one path is completely traversed or if the distance is now bigger than 1, the loop is completed and both pointers point at the furthest tile from the central tile where the distance is at most 1. Then the function `connect` establishes the necessary path in order to join both remaining parts of the paths in the shortest way: it is a finite selection of cases in each of which the construction is easy. This point raises no difficulty.

The procedure `measure` is detailed by Algorithm 3. It takes into account that the leftmost path which joins the central cell to the rightmost tile may lie on the left-hand side of the path joining the leftmost tile. This is why we have this selection of cases into three of them denoted by `equal`, when the distance is 0, `normal` when the distance is 1 and the tile on `Lmark` is on the left-hand side of the tile on `Rmark`, and `opposite` when the distance is 1 and the tile on `Lmark` is on the right-hand side of the tile on `Rmark`. Such cases do happen. Note that the case `opposite` is structurally very similar to the case `normal`. The only

difference is that in the former case, we look at `Rmark.status` while in the latter we look at `Lmark.status`. The reason is that in both cases, we have to consider the status of the rightmost tile.

Algorithm 2 *The computation of the path from the root to a node of the tree. shortest path. Given* `tile` = (`tile`(0),`tile`(1)), *the number of the sector and the number of the node respectively. Also,* `representation` = [`tile`(1)] *is an array of* 0*'s and* 1*'s indexed from* 0. *The index* `cursor` *starts from* 1 *and it is on the lowest digit.*

```
stage := new T_pairs;
stage.next := thepath;
stage.ingate := 1;
stage.outgate := 0; -- characterizes an end
thepath := letape;
while cursor < representation'last
loop
   stage := new T_pairs;
   stage.next := thepath;
   stage.ingate := 1;
   if cursor+1 > representation'last
     then
        stage.outgate := 4 + representation(cursor);
     else
        stage.outgate := 4 - representation(cursor+1)
                          + representation(cursor);
        if representation(cursor+1) = 1
          then if cursor+2 <= representation'last
                  then representation(cursor+2) := 1;
               end if;
        end if;
   end if;
   thepath := letape;
   cursor := cursor+2;
end loop;
--  finalization :
stage := new T_pairs;
-- inversion
stage.sortie := tile(0);
stage.entree := 0;
stage.next := thepath;
thepath := stage;
return thepath;
```

This allows us to briefly mention that the function `leftmost` works in a similar way, also based on the computation of the distance between the given path and the constructed one which should be the leftmost one. During the construction, the algorithm tries to keep the distance between the current tile on the given path and the current tile of the constructed path exactly equal to 1. As long as this is possible, the algorithm goes on in this way. If it can hold the condition until the last connection to the target of the path, we are done. But

it may happen, and this indeed does happen, that at the next step after the current path, the distance must be at least 2. This means that the constructed path went to much to the left and that it is needed to resume the computation

Algorithm 3 *The computation of the distance between two tiles, assuming that the distance between their fathers is at most* 1. *Given* `Lmark`,`Rmark`, *two pointers on the considered paths.*

```
distance0 := distance;
case side is
when equal => -- distance = 0
   if Lmark.ingate = 0 then -- initialisation
     maxim := maxi(Lmark.outgate,Rmark.outgate);
     minim := mini(Lmark.outgate,Rmark.outgate);
     ecart1 := maxim - minim;
     ecart2 := minim + 7 - maxim;
     ecart := mini(ecart1,ecart2);
   else -- ordinary situation: distance0 = 0
     if (Lmark.outgate /= 0) and (Rmark.outgate /= 0) then
       distance := maxi(Lmark.outgate,Rmark.outgate)
                   - mini(Lmark.outgate,Rmark.outgate);
     else distance := distance0;
     end if;
   end if;
   if Lmark.outgate /= Rmark.outgate then
     if Lmark.outgate = theleftmost(Lmark.outgate,
                                    Rmark.outgate) then
       side := normal;
     else side := opposite;
     end if;
   end if;
when normal => -- distance0 = 1
   if (Lmark.outgate /= 0) and (Rmark.outgate /= 0) then
     distance := 5 - Lmark.outgate + Rmark.outgate - 3;
     if Rmark.status = white then
       distance := distance+1;
     end if;
     if distance = 0 then side := equal; end if;
   else distance := distance0;
   end if;
when inverse =>
   if (Lmark.outgate /= 0) and (Rmark.outgate /= 0) then
     distance := 5 - Lmark.outgate + Rmark.outgate - 3;
     if Lmark.status = white then
       distance := distance+1;
     end if;
     if distance = 0 then side := equal; end if;
   else distance := distance0;
   end if;
end case;
```

from a previously reached tile. If it were needed to go back to the central time each time the computation has to be resumed, the algorithm would be quadratic.

Fortunately, a careful analysis of the construction shows that it is possible to find key tiles so that each time we have to resume the computation, we have to go back to the last fixed key tiles so that adding all the traversed parts of the path lead to a total length which is at most twice the length of the given path. The length of this part of the program exceeds the room for this paper. However, the overall computation of the function `leftmost` is still linear in time with respect to the length of the given path.

As a last auxiliary function, we just remind the algorithm for the Poisson generator, taking from [2]. We can rewrite it as follows:

Algorithm 4 *The Poisson generator for an integer valued function. Here,* `random` *is a uniform random integer-valued variable in the range* 0..`p_rand`. *The function* `random` *is an algorithmic random generator. It can be constructed as indicated also in* [2].

```
      compte : integer := 0;
      pois   : double := 1.0;
   begin
      loop
         pois := pois*double(random)/double(p_rand);
         exit when pois < exp(-lambda);
         compte := compte+1;
      end loop;
      return compte;
   end;
```

5.3 Implementing the protocol

The simulation is controlled by the procedure `execute`, see Algorithm 5. As there are many variables to collect partial results for later analysis, we use records in order to make the program more readable. The partial results concern the various kinds of messages, so that the fields of the records are `public`, `reply`, `write`, `nonpublic`, `erase`.

The function `init_config` initializes the table `space`. In particular, for each tile, it provides the information about the number of the tile, its sector, as well as the similar information for its seven neighbours. It also computes the table `associates` of the numbers of the sides of tile in the neighbour sharing this side.

In Algorithm 5, note that we perform addition on records thanks to the facility given by *ADA* to overload the usual signs of operation '+', '−', '×' as well as relations, '=' and '>=' as an example.

We develop the function `transition` in Algorithm 6.

The actual content of the function `transition` is performed by the procedure `action_in` which implements the exact choices of the simulation, see Algorithm 7.

As indicated in Section 4, public messages are sent and conveyed at odd times. Due to the copying process from`stack0` onto `stack1` on which the execution is based, the information about a public message has to be copied on `stack1` at even times: otherwise, the message would be erased. This is what the procedure `replicate` performs. The same procedure is also used for the erasing messages during the delay they observe at the tile which emitted a public message. Of course, at each replication, the delay is decreased by 1, until the delay reaches 1. At this moment, it is sent to catch the emitted message.

Algorithm 5 *The procedure* `execute`.

```
auxil := init_config;
collect(auxil,0,collect_at_t,nb_max_msg_t);
totals := collect_at_t;
the_max_msg := nb_max_msg_t;
themax_at_t := collect_at_t;
for t in 1..duration
loop
   space := transition(auxil,t);
   auxil := space;
   collect(auxil,t,collect_at_t,nb_max_msg_t);
   totals := totals + collect_at_t;
   themax_at_t := maxi(themax_at_t,collect_at_t);
   the_max_msg := maxi(the_max_msg,nb_max_msg_t);
end loop;
```

Algorithm 6 *The function* `transition`.

```
-- central tile:
action_in(0,0);
-- the other tiles:
for sect in 1..space'last(1)
loop
   for i in 1..space'last(2)
   loop
      action_in(sect,i);
   end loop;
end loop;
return copy(new_space);
```

Let us describe this procedure with more details. Let μ be a public message issued at time $2t-1$, with $t > 0$ by the procedure `send`. At the same time, the procedure `send` creates an erasing message μ_e with the same number as μ. In `send`, the Poisson random generator is called with the parameter `poisson_radius` in order to define the radius of propagation of μ. This initializes the field `wait` attached to μ_0. By construction, the radius is always positive. Note that there is no condition on the time for the management of an erasing message. As long as its delay is greater than 1, the erasing message remains in the tile T where

it was created. When the delay is 1, it is sent to the neighbours with its field `wait` set to 0. The procedure `convey` transmits the erasing message as its delay is now always 0, called at each time by `action_in`, see Algorithm 7.

Algorithm 7 *The procedure* `action_in`.

```
cursor := space(sect,i)(0).message_stack;
while (cursor /= null)
loop
   case cursor.the_type is
   when public =>
      if (time mod 2) /= 0 then
        if cursor.direct = null then
          send(from => (sect,i), cursor => cursor);
        else convey (from => (sect,i), cursor => cursor);
        end if;
      else -- even time:
        replicate(cursor, ontile => tile1);
        if poissonrandom(poisson_reply) > 0 then
          reply(from => (sect,i), place => cursor,
                        num => cursor.number);
        end if;
      end if;
   when nonpublic =>
      convey (from => (sect,i), cursor => cursor);
   when erasing => -- cursor.direct = null, always
      if cursor.wait = 1 then
        send(from => (sect,i), cursor => cursor);
      elsif cursor.wait = 0 then
        convey (from => (sect,i), cursor => cursor);
      else -- cursor.wait > 1
        replicate(cursor, ontile => tile1);
        tile1.last_message.wait := cursor.wait-1;
      end if;
   end case;
   cursor := cursor.next;
   compte := compte+1;
end loop;
if poissonrandom(poisson_write) > 0 then
  write (from => (sect,i));
end if;
if (time mod 2) = 0 then
  if poissonrandom(poisson_public) > 0 then
    init_cell(sect,i,new_space,public);
  end if;
end if;
```

It is not difficult to see that if μ is sent at the time t and μ_e is sent at the time $t+r$, then μ and μ_e are at the same tile at the time $t+2r$.

More precisely, μ and μ_e are present on the same tile at an even time: $t+2r$ has the same parity as t which is odd and when μ is sent, but μ_e is sent when

`wait` = 1. And so, the coincidence is detected by the procedure `convey` at the time $t+2r+1$: at this time, μ, which reached the tile τ at $t+2r$ is still there at $t+2r+1$. The situation is given in a schematic way by Table 8 and it is illustrated by Figure 5.

Algorithm 8 *The procedure* `action_in`.

```
--
-- sector      1   1   1       5   5   5
-- tile        7   3   1   0   1   9  24
-- wait        0   1   2   3   4   5   6
--
--        0    X
--        1        X                                   6
--        2        X                                   5
--        3            X                               4
--        4            X                               3
--        5                X                           2
--        6    o           X                           1
--        7        o               X                   0
--        8            o           X                   0
--        9                    o       X               0
--       10                        o   X               0
--       11                            o   X           0
--       12                                Z           0
--
```

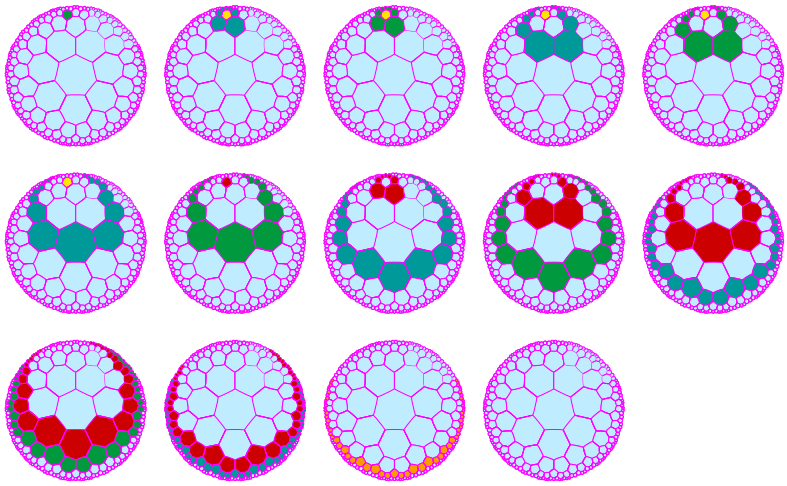

Figure 5 *The emission of a public message and of its erasing signal.*

In the program, the cancellation of a message reached by its erasing signal is performed by the procedure `convey`. When dealing with a public message, the first task of the procedure is to scan the stack in order to possibly detect an erasing signal with the same number as the message. When this happens, the message, which is scanned in `stack0` is simply not copied onto `stack1`. When the procedure examine an erasing signal, it performs a similar scanning: if a public message bears the same number as the signal, the signal is not copied onto `stack1`. As the cancellation has to occur when both signals are present, this simple way is enough to perform this action and there is no need to connect the two decisions of not copying the information onto `stack1`. This is guaranteed by the disjunction between `stack0` and `stack1`.

6 The experiment

The experiment was performed by the running the program on a simple laptop. The laptop is a `Lenovo` one, with two Intel processors, both working at 2 GHz, and Linux Mandriva as operating system. The used *ADA*-compiler belongs to the `gnu`-family, version 4.4.1. In the first sub-section, we describe the experiment and we give an account of the results. In the second sub-section, we give an interpretation of the results.

6.1 Description of the experiment and of the results

The program was run for six value of the depth of the Fibonacci tree, ranging from 5 to 10. Denote by $\mathcal{S}$ the observation space. The size of $\mathcal{S}$ is defined by the depth of the Fibonacci tree which spans the seven sectors displayed around the central cell. We consider the tiles whose level in the tree is at most `depth` which takes values in [5..10] in our experiments. This means that $\mathcal{S}$ contains 1625 tiles when `depth` = 5 and 200593 tiles when `depth` = 10. Increasing the depth by 1 means multiplying the number of tiles by a coefficient which quickly tends to $\frac{3+\sqrt{5}}{2} \approx 2.618034$. This factor of a bit more than 2.6, can be observed in Table 5 which indicates the number of tiles of $\mathcal{S}$ for the different values of `depth`.

The second parameter which we also changed is the radius of propagation of the public messages. As indicated in Section 4, the radius is an integer valued random variable following a Poisson law. The coefficient is fixed to 5 in one series of experiments and to 10 in the second one. The range of the variable is in mean this coefficient. However, the value may range from 0 to twice the coefficient with, from time to time, bigger values. Table 5 indicates the number of messages emitted in the observed area under these conditions. The remaining parameters are the following. Each tile is given the possibility to emit a message. Again we assume that the probability of such an event is given by a Poisson law. The coefficient is 0,005 for a public message, taking into account that such messages are emitted at odd times only. Also, the tiles which are on the border of the

space and which are the more numerous, more than 60% of the overall number of tiles in $\mathcal{S}$, are given an additional possibility with again a Poisson law whose coefficient is 0.0025. This is also to reflect the possibility for the tiles of $\mathcal{S}$ to receive messages from outside $\mathcal{S}$. Now, for private messages, they are caused either as a reply to a public message, or as a single message sent to a particular tile via the consultation of the directory. These events are also following a Poisson law and the coefficient is 0.0025 for the reply to a public message, and 0.001 for the consultation of a directory. In the case of a message sent after consulting the directory, it is assumed that the coordinate of the tile given by the directory falls within $\mathcal{S}$ and that both numbers constituting the coordinate are uniformly distributed in their respective ranges.

Table 5 *The number of tiles and the total number of messages. The number after* `sent` *is the radius of propagation. The time lines indicate the number of iterations during which the program was executed. Under the line indicating the time for radius* 5*, we indicate the ratio between the number of messages for consecutive depths as long as the overall duration is the same. The lines* `mean` *indicate the mean of the number of messages up to* t *divided by* t *for* $t \in [1..T]$ *where* T *is indicated by the line* `time`*.*

depth	5	6	7	8	9	10
tiles	1625	4264	11173	29261	76616	200593
sent, 5	1101	3308	8636	21797	49295*	60453*
time, 5	168	168	168	168	142	69
ratio		3.00454	2.61064	2.52396		
mean, 5	6.81949	19.13115	50.10053	128.31127	342.79960	877.83673
ratio		2.80536	2.61879	2.56108	2.67162	2.56079
sent, 10	2173	8289	13687*	13784*	19167*	30164*
time, 10	168	168	92	41	30	24
mean, 10	11.21332	40.57043	101.35430	197.08219	405.77815	965.53752
ratio		3.618057	2.50356	1.94449	2.058929	2.37947

Table 6 *The number of tiles and the total number of messages at time* 24*. The conventions are those of Table* 5*. Under each line indicating the number of messages sent at time* 24*, we have the ratio between two consecutive numbers.*

depth	5	6	7	8	9	10
tiles	1625	4264	11173	29261	76616	200593
sent, 5	169	423	1158	3017	7888	20556
ratio		2.50296	2.73759	2.60535	2.61452	2.60599
sent, 10	204	582	1538	4509	11413	30164
ratio		2.85294	2.64261	2.94129	2.53116	2.64295

Another important feature regarding private messages is that in the experiment, it was assumed that once a communication has started it goes on endlessly:

if A replies to a message sent by B, either public or private, then B replies to A which again replies to B and this process goes on periodically. Moreover, the reply was always assumed to be immediate.

Table 5 indicates the number of messages issued during the whole time of the simulation, measured by the number of iterations of the procedure `execute`. The number of iterations is also given by the table. It can be noticed that this number is always 168 for small values. This number was fixed for the experiment and it can be noticed that $168 = 7 \times 24$. For a fixed value of the mean radius of propagation, we notice that the number of iteration becomes lower and lower. This is a limitation caused by the system and the machine under which the program was run. It can be noticed that the decay of the number of the iterations corresponds to the increase of the number of tiles. Accordingly, this alters the number of messages which were issued.

Table 5 indicates the ratio between the numbers of messages sent when the radius of propagation is 5 and when the depth of the Fibonacci tree is 5, 6 and 7 as for these depths, we have the same duration 168. In Table 6, we indicate the overall number of sent messages at time 24 as we have data for each depth of the experiment. This allows us to compute the ratio between numbers associated to consecutive depths.

Table 7 *The number of tiles and the maximal number of messages passing through a tile at a time during the interval of observation. The number after* `max` *is the radius of propagation. The rest of the conventions are those of Table* 5.

depth	5	6	7	8	9	10
tiles	1625	4264	11173	29261	76616	200593
max, 5	18	39	58	104	232*	192*
ratio		2.16667	1.48718	1.79310		
time, 5	168	168	168	168	142	69
max, 10	63	169	204*	197*	315*	694*
time, 10	168	168	92	41	30	24

Table 8 *The number of tiles and the maximal number of messages at time* 24. *The conventions are those of Table* 5. *Under each line indicating the maximal number of messages passing through a tile at time* 24, *we have the ratio between two consecutive numbers.*

depth	5	6	7	8	9	10
tiles	1625	4264	11173	29261	76616	200593
sent, 5	11	16	25	34	54	91
ratio		1.45455	1.56250	1.36000	1.58824	1.68519
sent, 10	17	30	61	140	315	694
ratio		1.76471	2.03333	2.295082	2.25000	2.20317

Table 5 also reports another measurement performed by the program: if n_t is the number of messages emitted up to the time t with $t \in [1..T]$, where T is the duration of the experiment, *i.e.* the number of iterations, then the lines `mean` give the mean value of the numbers $\frac{n_t}{t}$. These values are computed when the propagation radius is 5 and when it is 10.

Another interesting information is the maximal number of messages passing through a tile. The data are given in Table 7, in the same conditions as in Table 5.

The data are summarized in Tables 7 and 8.

Below, Tables 9 and 10 give an information on the decomposition of the number of emitted messages between the public messages and the private ones

Table 9 *This table is a refinement of Table* 5. *It indicates how the number of messages emitted in* $\mathcal{S}$ *are distributed between the public and the private messages and, among the latter ones, between the replies to a public message or a direct message to a single tile via the directory. This table gives the data for radiuses* 5 *and* 10 *for the propagation of the public messages, upper and lower halves of the table, respectively.*

depth	time	public	reply	write	total
5	168	636	211	254	1101
ratio		0.577	0.192	0.231	
6	168	1840	783	685	3308
ratio		0.556	0.237	0.207	
7	168	4669	2285	1682	8636
ratio		0.541	0.264	0.195	
8	168	11982	5295	4520	21797
ratio		0.550	0.243	0.207	
9	142	27099	12488	9708	49295
ratio		0.550	0.253	0.197	
10	69	34536	13467	12450	60453
ratio		0.571	0.223	0.206	
5	168	654	1281	238	2173
ratio		0.301	0.589	0.110	
6	168	1791	5848	650	8289
ratio		0.217	0.705	0.078	
7	92	2472	10279	936	13687
ratio		0.181	0.751	0.068	
8	41	3094	9603	1087	13784
ratio		0.224	0.697	0.079	
9	30	6107	10937	2123	19167
ratio		0.318	0.571	0.111	
10	24	12935	12954	4275	30164
ratio		0.429	0.429	0.142	

and, among the latter, between replies to a public message and direct messages to a tile whose coordinates are delivered by the directory.

As for Table 5, Table 9 gives the information for each area of $\mathcal{S}$ defined by the depth of the Fibonacci tree. The upper half of the table concerns a propagation of the public messages characterized by radius 5, while, the lower half concerns radius 10.

Table 10 *This table is a refinement of Table* 6. *It indicates how the number of messages emitted in* $\mathcal{S}$ *are distributed between the public and the private messages and, among the latter ones, between the replies to a public message or a direct message to a single tile via the directory. All data are taken at iteration* 24. *In the upper half of the table, the radius of propagation for the public messages is* 5. *In the lower half, the radius is* 10.

depth	public	reply	write	total
5	109	24	36	169
ratio	0.577	0.192	0.231	
6	277	53	93	423
ratio	0.556	0.237	0.207	
7	738	200	220	1158
ratio	0.541	0.264	0.195	
8	1856	504	657	3017
ratio	0.550	0.243	0.207	
9	5021	1268	1599	7888
ratio	0.550	0.253	0.197	
10	13026	3181	4349	20556
ratio	0.571	0.223	0.206	
5	98	78	28	204
ratio	0.480	0.382	0.138	
6	273	205	104	582
ratio	0.469	0.352	0.179	
7	672	614	252	1538
ratio	0.437	0.399	0.164	
8	1919	1945	645	4509
ratio	0.426	0.431	0.143	
9	5010	4720	1683	11413
ratio	0.439	0.414	0.147	
10	12935	12954	4275	30164
ratio	0.429	0.429	0.142	

In Table 10, the data are attached to the same time, defined by 24 iterations. This gives a direct comparison between all the data but it does not concern a long enough time period.

We turn to the next sub-section where we try to extract a general information from these data and from a few other ones we have not the room to give in this paper.

6.2 Interpretation

Several conclusions can be drawn from the results presented in Subsection 6.1.

The first one concerns the ratios between the number of messages for consecutive depths of the Fibonacci trees. These ratios are close to the ratio between the area of $\mathcal{S}$ for consecutive values of the depth of the spanning tree. It seems that we may conclude that these experimental data support Assumption 1.

Assumption 1 *For any t, the number of messages issued at the time t in $\mathcal{S}$ is proportional to the number of tiles belonging to $\mathcal{S}$.*

Indeed, the coefficient of the Poisson law is a kind of mean of the random variable indicating whether a message is sent or not. According to the homogeneous nature of the space and as the decision of one tile is independent from that of its neighbours, it can be expected that the observed number of issued messages is proportional to the area. This conclusion is strengthened by the following consideration. As the actual radius of propagation of the public messages is bounded, the contribution of far tiles is ruled out, starting from a certain distance from a tile and this distance can be uniformly bounded for all the tiles. The values of the radius exceeding say twice the mean of the radius as a random variable which we assumed to follow a Poisson law can be considered as an event of very small probability so that an infinite repetition of exceptionally long radiuses can be considered as an event of probability 0. The same remark apply if we relax a bit the conditions on the coordinates provided by the directory. We may assume that the number of the sector is uniformly distributed and that the number of the tile in the tree is an integral random variable following a Poisson law with a radius of the same size as that of $\mathcal{S}$. And so, relaxing a bit the condition on the directory as just suggested does not alter the argument in favor of Assumption 1.

No clear statement can be inferred from Tables 7 and 8, except the fact that the maximal number of messages passing through $\mathcal{S}$ seems to be increasing with the time. This can also be seen on the experiment for each depth: as the number of iteration increases, the maximal number of messages passing at a tile also increases.

It is also interesting to look at where the maximal number of messages appear. We have not the room to give the relevant information computed by the program. For each iteration, the program indicates a tile at which the number of passing messages is maximal. It also splits the information in looking at where is obtained the maximal number of passing public messages or private messages replying to a public message or private messages written to a single tile. It is interesting to notice that most often the tiles are not the same for the different kinds of messages. Also, the position of this maximum is generally the central tile or one

of its neighbours. However, for the emission of the pubic message, the maximal number of passages at a tile may be obtained a bit further from the central tile, while the replies to these messages seem to be maximal most often at the central tile or its immediate neighbours.

Tables 9 and 10 show a very interesting difference between the cases when the radius of propagation of the public messages is 5 and when it is 10. In both tables, we can see that the proportion of public messages is higher when the radius is 5. This is particularly striking in Table 9, but it is already noticeable in Table 10 showing that this difference appears quickly and that it tends to increase a bit with the time. It is also interesting to see that the relative 'loss' of the public messages 'benefit' to their replies. Indeed, in both tables, the proportion of direct messages to a single tile is not improved when the depth of the Fibonacci tree is increasing. Of course, the Poisson coefficient for triggering a public message is 0.005, while that of a reply is 0.0025 and that of a direct private message is 0.001. However, the public messages are triggered at odd times only and the replies occur only at even time while the direct private messages can be sent at any time. There should be no big difference between direct private messages and replies to a public message. In fact the explanation lies in the geometry of the space. Indeed, the reply is proposed to any tile visited by the propagation wave which covers all the tiles within the radius fixed at the time when the public message was emitted. This additional solicitation explains the importance of the replies.

At this point, we can also indicate why we have chosen a Poisson law to model this message system. The reason is that we have to take into account the geometry of the space. A uniform distribution would give much more weight to distant tiles by the simple fact that their number increases exponentially: the farther they are the more message they would send to the centre. This is also the reason why we decided to limit the propagation of the public messages. If no limitation would be put, the number of messages received at any point would grow exponentially with time by the just mentioned argument. Accordingly, the limitation restricts this possibility. Now, the Poisson law also gives the possibility to obtain big values with respect to the mean value. Simply these extremal are very rare, the more rare they are higher values.

7 Conclusion

It is the place here to discuss how these results can be interpreted in a more qualitative way. The number of iterations suggests that the unit of time is an hour. The tiles can be interpreted either as individuals or as groups of individuals in a given constant area, the one defined by the area of a tile. Remember that in the space we consider, all tiles have the same area. The limitation of the public messages can be interpreted as a natural limit due to the conditions in which the message is sent and also depending on the intentions of the sender.

Two important points should be noted. The first one is the property of the public messages to cover all the tiles of a given area for each tile to receive the

message once exactly. The second point is the mechanism to limit the propagation of a public message. This mechanism needs no centralization. It is monitored by the sender and, a priori, each tile can be a sender. To be a sender is defined by a probability which is the same for every tile. The third point is that in some sense, the indicated scenario is a worse case one with respect to the traffic load supported by each tile. Indeed, the fact that once a communication is established between two tiles goes on endlessly contributes to increase the traffic with the time. There is room here to tune the modeling by introducing various ways to delay answers or to limit the number of contacts of a tile with others: here also we could consider that this number is a Poisson random variable whose mean can be fixed uniformly or depending on other criteria which we have not considered here. A last point is the possible improvement of the program in order to obtain more data and to go further in the exploration of the simulation space. Note that the file which records all the communications when the depth is 5 and the radius of propagation is 10 and the number of iterations is 168 has a size of around 164 megabytes. As mentioned in Section 6, increasing the depth by 1 multiplies the area by around 2.618. Accordingly, the depth which defines the area of $\mathcal{S}$ cannot be extent very much. Already depth 10 with radius 10 requires a machine more powerful than a simple laptop.

We are convinced that there is further work ahead to better analyze the data already obtained, to improve the program in order to go further in the exploration of the simulation space. There is also room to tune the basic parameters in order to get a picture closer to real networks as, for instance, social networks.

References

1. R. Bonola, *Non-Euclidean Geometry*, *Dover*, 0-486-60027-0, New-York, 389p.
2. D. Knuth, *The Art of Computer Programming, Volume* 2*, Seminumerical Algorithms*, *Addison-Wesley*, Reading, Massachusetts, (1997), third edition, 724p.
3. M. Margenstern, Implementing Cellular Automata on the Triangular Grids of the Hyperbolic Plane for New Simulation Tools, **ASTC'2003**, (2003), Orlando, March, 29- April, 4.
4. M. Margenstern, On the communication between cells of a cellular automaton on the penta- and heptagrids of the hyperbolic plane, *Journal of Cellular Automata* **1**(3), (2006), 213-232.
5. M. Margenstern, *Cellular Automata in Hyperbolic Spaces, Volume* 1*, Theory*, *OCP*, Philadelphia, (2007), 422p.
6. M. Margenstern, *Cellular Automata in Hyperbolic Spaces, Volume* 2*, Implementation and computations*, *OCP*, Philadelphia, (2008), 360p.
7. M. Margenstern, Possible Applications of Navigation Tools in Tilings of the Hyperbolic Plane, *Lecture Notes in Electrical Engineering*, **70**, (2011), 217-229.